情報処理学会論文誌 Vol.59 No.1 1–3 (Jan. 2018)

論文タイトル

吉田 昂太

受付日 xxxx年0月0日, 採録日 xxxx年0月0日

Kota Yoshida

Received: xx 0, xxxx, Accepted: xx 0, xxxx

1. 概要

現在の多くのプロセッサは投機的実行を悪用する Spectre 攻撃に対して脆弱である。これらの攻撃に対処するソフトウェアベースの方法として、プログラム中から Spectre 攻撃に対して脆弱なコード辺 (Spectre ガジェット) を特定し、部分的に投機的実行を抑制する方法がある。既存研究では Spectre ガジェットを検出する方法として記号実行を用いる手法が提案されているが、通常の実行パスと投機的実行パスの両方を探索する必要があるため、探索する状態空間が非常に多くなり複雑なプログラムに対してスケールしない問題がある。本論文では、Spectre ガジェットの検出確率が低い投機的な状態の探索を避けることで、記号実行のスケーラビリティを向上させる手法を提案する。また、記号実行において探索されなかった投機的状態はファジングを用いて探索することで、スケーラビリティと精度の両立を目指す。

2. はじめに

3. 背景

3.1 投機実行

現代のほぼ全ての CPU は 1 つの命令を、命令フェッチ、デコード、実行などの複数のステージに分割して実行するパイプライン方式を採用している。このようにすることで、前の命令が全ての処理を終えることを待たずに、次の命令の処理を開始できる。このように複数の命令を並行して処理することで CPU はスループットを向上させている。しかし、次に実行すべき命令が前の命令の実行結果に依存している場合、CPU は次にどの命令を実行するべきかわからないため命令の処理を停止させる必要がある。このような状況を制御ハザードと呼び、CPU のパフォーマンスが大幅に低下する可能性がある。このような制御ハザードによるストールを回避するため、現代の CPU は分岐命令に遭遇した場合、分岐予測器によって分岐先を予測し、後続の命令を投機的に実行する。(TODO: 分岐予測器の図と説明)。投機的に実行された命令と実行結果は CPU 内部の Reorder Buffer (ROB) という機構で管理され、依存先の命令の完了を待機する。そのため、投機的に実行できる命令数は ROB の大きさに制限されており、マイクロオペレーション (μOP) で 200 命令程度である (TODO: ソースは？)。分岐先が確定し予測が正しかった場合は、投機的実行の結果をレジスタやメモリなどのハードウェア状態に反映させる。分岐先が誤っていた場合は ROB から投機的実行の結果を破棄し、誤った予測が行われた時点のアーキテクチャ状態までロールバックされる。

3.2 Side and Covert Channels

コンピュータのシステムにおいて、channel とは情報を送信する可能性のある媒体のことを言う。channel には大きく分けて legitimate channel と incidental channel の 2 種類が存在する。legitimate channel は システムの設計者が情報送信用に意図したチャネルであり、イーサネット、共有メモリ、IPC ソケットなどがある。逆に incidental channel は 偶発的に設計されたチャネルであり、リソースの競合、CPU キャッシュの状態、電力消費の変化などがある。更に、セキュリティ脅威モデルのコンテキストにおいて incidental channel は covert channel と side channel の 2 種類に分類される。covert channel は悪意のある送信者と受信者が意図的に情報の伝達を行うために利用される channel である。一方で side channel は、送信者は受信者に情報を伝達することを意図しておらず、情報が悪意のある受信者に伝達 (つまり漏洩) される際に利用される channel である。つまり side channel を考慮する場合、情報を送信する被害者と情報を受け取る攻撃者が存在する。incidental channel は 情報を伝達する方法に基づいて、タイミングベース、アクセスベース、またはトレースベースの channel に分類できる。タイミングベースの channel は、さまざまな操作のタイミングを利用して情報を漏洩します (例: [6、11、40])。たとえば、1 つのプロセスが多数のメモリ アクセスを実行すると、別のプロセスのメモリアクセスが遅くなります。

3.3 一時実行攻撃

一時実行攻撃とは、CPU の投機的実行によって一時的に実行される命令の結果をマイクロアーキテクチャに痕跡を残すことを利用する攻撃法である。本来、CPU は誤った投機的実行が行われた場合、その結果はマイクロアーキテクチャの状態には反映されず、パイプラインはフラッシュされる。しかし、キャッシュなどの一部のマイクロアーキテクチャの状態はパフォーマンスの観点からそのまま維持される。これを利用して、攻撃者は投機的実行を誘発させ、マイクロアーキテクチャの状態を通じて、後から秘密情報などを復元することが可能である。一時実行攻撃は 2018 年に Spectre 攻撃 [3] と Meltdown 攻撃 [4] が初めて明らかにされて以来、様々な CPU を標的とした、多数の新しい一時実行攻撃が発見されてきた。これらの攻撃は大きく分けて Spectre 型と Meltdown 型に分類される [5]。Spectre 型の攻撃 [3, 6–8] はデータフローまたは制御フローの予測ミスに続く一時的な命令を悪用する。一方で, Meltdown 型の攻撃 [**?**, 4, 9–11] は fault を発生させる命令に続く一時的な命令を悪用する。一時実行攻撃は大きく分けて 3 つのフェーズで構成される。(TODO: 概要図)。まず攻撃者は分岐予測器やデータキャッシュの状態を設定し、マイクロアーキテクチャを目的の状態にする。次に、投機実行を引き起こす命令を実行する。これは、例外や分岐予測ミスなどにより、後続の命令が最終的に潰されるような命令である。CPU はトリガー命令が完了する前に後続の命令を一時的に実行する。この一時実行命令はマイクロアーキテクチャの秘密チャネルの送信側として機能し、秘密に依存するメモリ位置を CPU キャッシュにロードしたりする。トリガー命令の処理を終了すると、CPU は例外または分岐予測ミスを検出し、パイプラインをフラッシュし、アーキテクチャの状態をロールバックする。最後に攻撃者は、秘密チャネルの受信側で、メモリアクセスのタイミングを計って、秘密情報を一時実行命令から推測するなどして、許可されていない一時実行命令の結果を復元する。

3.4 Spectre attack

本研究の対象となる Spectre-V1 の解説

3.5 記号実行

3.6 ファジング

ファジングは、ソフトウェアの欠陥や脆弱性を検出することを目的としたテスト手法である。ファジングは多数のテストケースを対象ソフトウェアへの入力として生成し、その実行結果を観測することでバグや脆弱性を検出する。単純にランダムにテストケースを生成すると入力空間が膨大になり非効率的であるため、多くのファジングツールは冗長なテストケースやバグを起こす可能性の低いテストケースの生成を回避する手法を用いている。

4. KLEESpectre

5. 問題設定

モチベ例について説明
SpecFuzz の結果を引用

6. 提案手法

7. 実装

8. 評価

9. 関連研究

10. 結論

参考文献

[1] Colin Percival. Cache missing for fun and profit, 2005.
[2] Yuval Yarom and Katrina Falkner. {FLUSH+RELOAD}: A high resolution, low noise, l3 cache {Side-Channel} attack. In *23rd USENIX security symposium (USENIX security 14)*, pp. 719–732, 2014.
[3] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael

Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1–19, 2019.

[4] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, pp. 973–990, Baltimore, MD, August 2018. USENIX Association.

[5] Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin Von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, and Daniel Gruss. A systematic evaluation of transient execution attacks and defenses. In *28th USENIX Security Symposium (USENIX Security 19)*, pp. 249–266, 2019.

[6] Esmaeil Mohammadian Koruyeh, Khaled N. Khasawneh, Chengyu Song, and Nael Abu-Ghazaleh. Spectre returns! speculation attacks using the return stack buffer. In *12th USENIX Workshop on Offensive Technologies (WOOT 18)*, Baltimore, MD, August 2018. USENIX Association.

[7] Giorgi Maisuradze and Christian Rossow. ret2spec: Speculative execution using return stack buffers. CCS '18, p. 2109–2122, New York, NY, USA, 2018. Association for Computing Machinery.

[8] Jann Horn. speculative execution, variant 4: speculative store bypass. 2018.

[9] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient {Out-of-Order} execution. In *27th USENIX Security Symposium (USENIX Security 18)*, pp. 991–1008, 2018.

[10] Stephan Van Schaik, Alyssa Milburn, Sebastian Österlund, Pietro Frigo, Giorgi Maisuradze, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Ridl: Rogue in-flight data load. In *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 88–105. IEEE, 2019.

[11] Jo Van Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lippi, Marina Minkin, Daniel Genkin, Yuval Yarom, Berk Sunar, Daniel Gruss, and Frank Piessens. Lvi: Hijacking transient execution through microarchitectural load value injection. In *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 54–72. IEEE, 2020.

[12] Atri Bhattacharyya, Alexandra Sandulescu, Matthias Neugschwandtner, Alessandro Sorniotti, Babak Falsafi, Mathias Payer, and Anil Kurmus. Smotherspectre: exploiting speculative execution through port contention. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 785–800, 2019.

[13] Michael Schwarz, Claudio Canella, Lukas Giner, and Daniel Gruss. Store-to-leak forwarding: leaking data on meltdown-resistant cpus (updated and extended version). *arXiv preprint arXiv:1905.05725*, 2019.

[14] Daniel Weber, Ahmad Ibrahim, Hamed Nemati, Michael Schwarz, and Christian Rossow. Osiris: Automated discovery of microarchitectural side channels. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 1415–1432, 2021.

[15] Daniel Moghimi, Moritz Lipp, Berk Sunar, and Michael Schwarz. Medusa: Microarchitectural data leakage via automated attack synthesis. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 1427–1444, 2020.

[16] Oleksii Oleksenko, Marco Guarnieri, Boris Köpf, and Mark Silberstein. Hide and seek with spectres: Efficient discovery of speculative information leaks with random testing. *arXiv preprint arXiv:2301.07642*, 2023.